

## ANSIBLE BEST PRACTICES - WINDOWS



CLASS PAGE

<https://jrueles.github.io/ansible-windows-best>

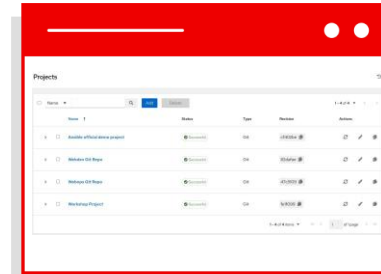


© 2025 by Innovation In Software Corporation

## PROJECT

A project is a logical collection of Ansible Playbooks, represented in Ansible Automation Controller.

You can manage Ansible Playbooks and playbook directories by placing them in a source code management system supported by Ansible Tower, including Git, Subversion, and Mercurial.



## PROJECT

- Under **Projects** in the left navigation bar, an example project named **Demo Project** is displayed.
- This project is configured to get Ansible project materials, including a playbook, from a public Git repository.
- You can also prepare a credential so you can access a private Git repository that needs authentication.

The screenshot shows a configuration window titled 'PROJECTS / Demo Project'. It has several tabs: 'DETAILS' (selected), 'PERMISSIONS', 'NOTIFICATIONS', 'JOB TEMPLATES', and 'SCHEDULE'. The 'DETAILS' tab contains the following fields:

- \* NAME: Demo Project
- DESCRIPTION: (empty)
- \* ORGANIZATION: Default
- \* SCM TYPE: Git
- SOURCE DETAILS:
  - \* SCM URL: https://github.com/ansible/ansible-tower.git
  - \* SCM BRANCH/TAG/COMMIT: (empty)
  - \* SCM CREDENTIAL: (empty)
- SCM UPDATE OPTIONS:
  - ☐ CLEAR
  - ☐ DELETE ON UPDATE
  - ☒ UPDATE REVISION ON LAUNCH
- CACHE TIMEOUT (SECONDS): 0

At the bottom right, there are 'CANCEL' and 'SAVE' buttons.

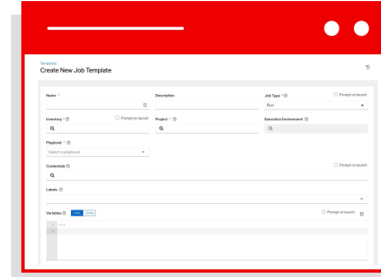
## JOB TEMPLATES

Everything in Ansible Tower revolves around the concept of a **Job Template**. Job Templates allow Ansible Playbooks to be controlled, delegated and scaled for an organization.

Job templates also encourage the reuse of Ansible Playbook content and collaboration between teams.

A **Job Template** requires:

- ▶ An **Inventory** to run the job against
- ▶ A **Credential** to login to devices.
- ▶ A **Project** which contains Ansible Playbooks

A screenshot of the 'Create New Job Template' form in Ansible Tower. The form is titled 'Create New Job Template' and has a red header bar. It contains several fields: 'Name' (text input), 'Description' (text area), 'Job Type' (dropdown menu), 'Inventory' (dropdown menu), 'Project' (dropdown menu), 'Credential' (dropdown menu), 'Playbook' (text input), 'Launch' (button), and 'Variables' (text area). There are also checkboxes for 'Run' and 'Refresh'.

## JOB TEMPLATES

- Under **Templates** in the left navigation bar, an example template called **Demo Job Template** is displayed.
- This job template runs the `hello_world.yml` playbook from **Demo Project** on the hosts in **Demo Inventory**, using **Demo Credential** to authenticate access.
- This initial job template can be used to test Ansible Tower.

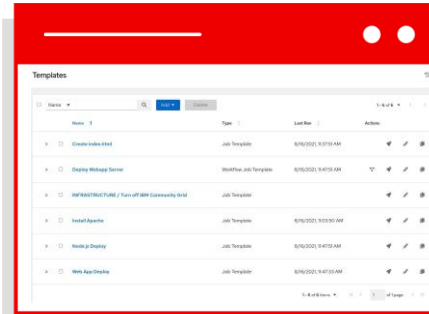
The screenshot shows the 'Demo Job Template' configuration page in Ansible Tower. The page has a top navigation bar with tabs: 'GENERAL', 'NOTIFICATIONS', 'COMPLETION', 'SCHEDULE', and 'ADVANCED'. The 'GENERAL' tab is active. The form is divided into several sections:

- NAME:** 'Demo Job Template' (read-only).
- DESCRIPTION:** A text area for describing the template.
- PLAYBOOK:** A dropdown menu showing 'hello\_world.yml'.
- PROJECT:** A dropdown menu showing 'Demo Project'.
- INVENTORY:** A dropdown menu showing 'Demo Inventory'.
- CREDENTIAL:** A dropdown menu showing 'Demo Credential'.
- ENVIRONMENT:** A dropdown menu for selecting an environment.
- WORKFLOW:** A dropdown menu for selecting a workflow.
- TIMEOUT:** A text input field for setting a timeout.
- OPTIONS:** A section with checkboxes for 'ENABLE PARALLEL EXECUTION', 'ALLOW PROCESSING ON HOSTS', and 'ENABLE CONCURRENCY'.
- EXTRA VARIABLES:** A text area for defining extra variables.

At the bottom right, there are buttons for 'SAVE', 'CANCEL', and 'DELETE'.

## EXPANDING JOB TEMPLATES


Job Templates can be found and created by clicking the **Templates** button under the *Resources* section on the left menu.

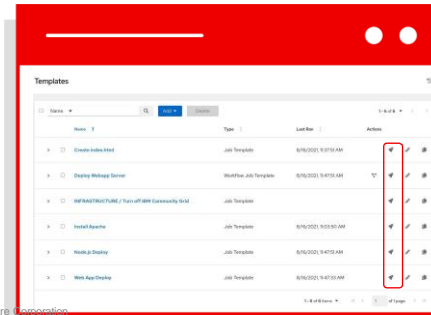


© 2025 by Innovation In Software Corporation

7

## EXECUTING AN EXISTING JOB

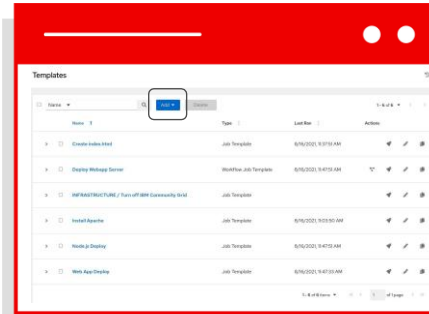
Job Templates can be launched by clicking the **rocketship button** for the corresponding Job Template 





## CREATING A NEW JOB TEMPLATE (1/2)

New Job Templates can be created by clicking the **Add** button



© 2025 by Innovation In Software Corporation

5

## CREATING A NEW JOB TEMPLATE (2/2)

This **New Job Template** window is where the inventory, project and credential are assigned. The red asterisk \* means the field is required.

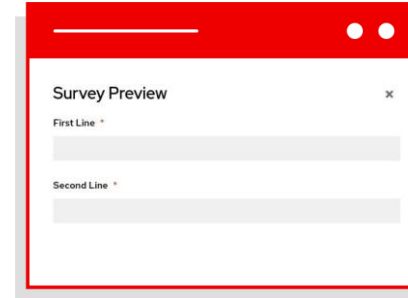
The screenshot shows a 'Create New Job Template' window. The window has a red title bar and a white content area. The content area contains several input fields with red asterisks indicating they are required: Name, Description, Job Type, Inventory, Project, Credential, and Labels. There are also dropdown menus for 'Select a job template' and 'Select a credential'. A 'Save' button is visible at the bottom left.

## SURVEYS

Controller surveys allow you to configure how a job runs via a series of questions, making it simple to customize your jobs in a user-friendly way.

An Ansible Controller survey is a simple question-and-answer form that allows users to customize their job runs.

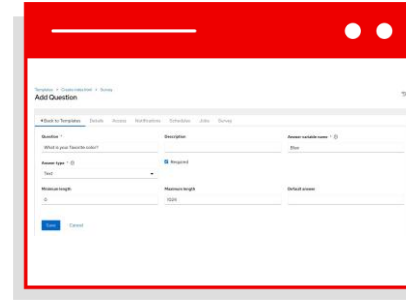
Combine that with Controller's role-based access control, and you can build simple, easy self-service for your users.

A screenshot of a 'Survey Preview' window. The window has a red title bar with two white circular buttons on the right. The main content area is white and contains the title 'Survey Preview' with a close button 'x' in the top right corner. Below the title, there are two text input fields. The first field is labeled 'First Line \*' and the second field is labeled 'Second Line \*'. Both fields are currently empty.

## CREATING A SURVEY (1/2)

Once a Job Template is saved, the Survey menu will have an **Add Button**

Click the button to open the Add Survey window.



The screenshot shows a window titled 'Add Question' with a red header bar. The window contains a form with the following fields and options:

- Question:** A text input field with the placeholder 'What is your favorite color?'.
- Description:** A text input field.
- Question variable name:** A text input field with the value 'Color'.
- Answer type:** A dropdown menu with 'Text' selected.
- Required:** A checkbox that is checked.
- Minimum length:** A text input field with the value '0'.
- Maximum length:** A text input field with the value '1000'.
- Default answer:** A text input field.

At the bottom of the form, there are two buttons: 'Add' (in blue) and 'Cancel'.

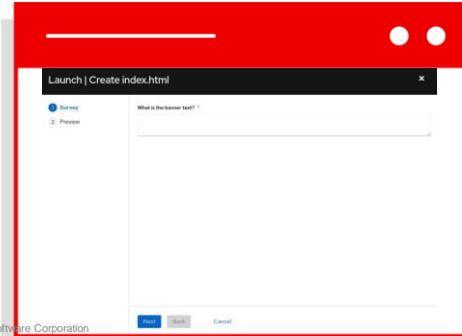
## CREATING A SURVEY (2/2)

The Add Survey window allows the Job Template to prompt users for one or more questions. The answers provided become variables for use in the Ansible Playbook.

The image displays two side-by-side screenshots of a web application interface, both featuring a red header bar with three white window control buttons (minimize, maximize, close). The left screenshot shows the 'Add Question' window, which has a breadcrumb trail 'Inventory > Hosts/Groups > Survey' and a tabbed interface with 'Add Question' selected. It contains fields for 'Question' (with a 'Description' link), 'Answer type' (set to 'Text'), 'Minimum length' (10), and 'Maximum length' (1000). The right screenshot shows the 'Survey' window, with a breadcrumb trail 'Inventory > Hosts/Groups > Survey' and a tabbed interface with 'Survey' selected. It includes a 'Survey' toggle switch, a 'Question' field, and tabs for 'Type', 'Variables', and 'Default'.

## USING A SURVEY

When launching a job, the user will now be prompted with the Survey. The user can be required to fill out the Survey before the Job Template will execute.

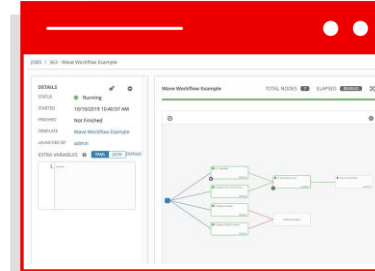


© 2025 by Innovation In Software Corporation

## WORKFLOWS

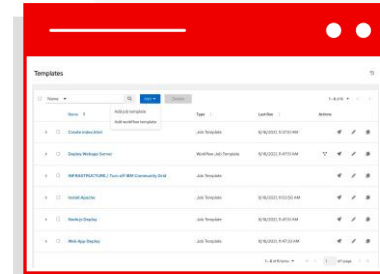
Combine automation to create  
something bigger

- ▶ Workflows enable the creation of powerful holistic automation, chaining together multiple pieces of automation and events.
- ▶ Simple logic inside these workflows can trigger automation depending on the success or failure of previous steps.



## ADDING A NEW TEMPLATE

▶ To add a new **Workflow** click on the **Add** button.  
This time select the **Add workflow template**




15

© 2025 by Innovation In Software Corporation



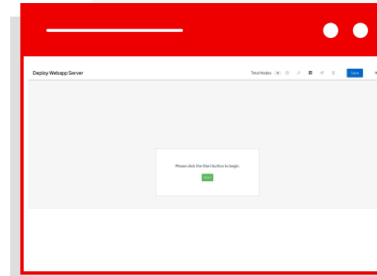
## CREATING THE WORKFLOW

Fill out the required parameters and click **Save**.  
As soon as the Workflow Template is saved the Workflow Visualizer will open.



## WORKFLOW VISUALIZER

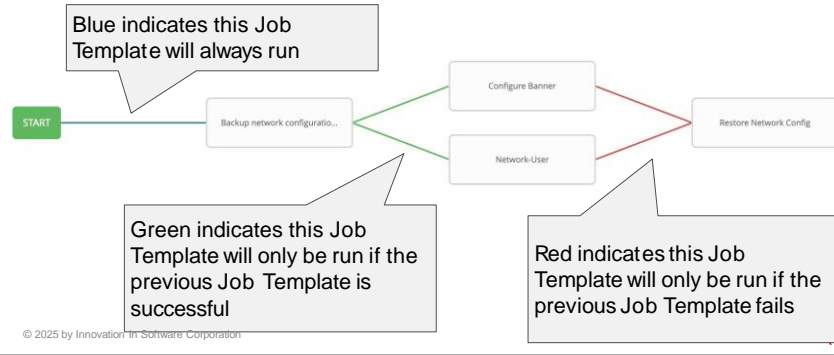
- ▶ The Workflow Visualizer will start as a blank canvas.
- ▶ Click the green Start button to start building the workflow.



© 2025 by Innovation In Software Corporation

## VISUALIZING A WORKFLOW

Workflows can branch out, or converge in.







## DEBUGGING ANSIBLE

© 2025 by Innovation In Software  
Corporation



## DEBUGGING ANSIBLE



Ansible offers a task debugger so you can fix errors during execution instead of editing your playbook and running it again to see if your change worked.

You have access to all the features of the debugger in the context of the task. You can check or set the value of variables, update module arguments, and re-run the task with the new variables and arguments.

The debugger lets you resolve the cause of the failure and continue with playbook execution.

## DEBUGGING ANSIBLE



For performance reasons, the debugger is not enabled by default. It must be enabled before playbook execution.

To enable the debugger:

Enable in configuration file:

```
[defaults]  
enable_task_debugger = True
```

Enable using environment variable:

```
ANSIBLE_ENABLE_TASK_DEBUGGER=True ansible-playbook playbook.yml
```



## DEBUGGING ANSIBLE



The debugger can be enabled/disabled at the task, block, role, or play level. This is especially useful when developing or extending playbooks, plays, and roles.

You can enable the debugger on new or updated tasks. If they fail, you can fix the errors efficiently.

## DEBUGGING ANSIBLE



The `debugger` keyword accepts five values.

Invoke the debugger:

- `always`:
  - Always, regardless of the outcome
- `never`:
  - Never, regardless of the outcome
- `on_failed`:
  - Only if a task fails
- `on_unreachable`:
  - Only if a host is unreachable
- `on_skipped`:
  - Only if the task is skipped

## DEBUGGING ANSIBLE



### Examples of using the debugger keyword (task)

```
tasks:
- name: Execute a command
  command: "false"
  debugger: on_failed
```

### Examples of using the debugger keyword (play)

```
- name: My play
  hosts: all
  debugger: on_skipped
  tasks:
    - name: Execute a command
      command: "true"
      when: False
```

## DEBUGGING ANSIBLE



Examples of setting the debugger keyword at multiple levels

```
- name: Play
  hosts: all
  debugger: never
  tasks:
    - name: Execute a command
      command: "false"
      debugger: on_failed
```

In this example, the debugger is set to `never` at the play level and to `on_failed` at the task level. If the task fails, Ansible invokes the debugger, because the definition on the task overrides the definition on its parent play.

DEBUGGER COMMANDS



Command	Shortcut	Action
print	p	Print information
task.args[key] = value	N/A	Update module arguments
task_vars[key] = value	N/A	Update task variables (you must update_task next)
update_task	u	Recreate a task with updated task variables
redo	r	Run the task again
continue	c	Continue executing, starting with the new task
quit	q	Quit the debugger

## DEBUGGING ANSIBLE



After Ansible invokes the debugger, use the commands to resolve the error that Ansible encountered.

Consider this example playbook, which defines the `var1` variable but uses the undefined `wrong_var` variable in a task by mistake.

```
- hosts: test
  debugger: on_failed
  vars:
    var1: value1
  tasks:
    - name: Use wrong variable
      ping: data={{ wrong_var }}
```

DEBUGGING ANSIBLE

If you run this playbook, Ansible invokes the debugger when the task fails.

From the debug prompt, you can change the module arguments or variables and run task again.

```
PLAY *****
TASK [wrong variable] *****
fatal: [192.0.2.10]: FAILED! => ("failed": true, "msg": "ERROR! 'wrong_var' is undefined")
Debugger invoked
[192.0.2.10] TASK: wrong variable (debug)> p result._result
{'failed': True,
 'msg': 'The task includes an option with an undefined variable. The error '
       'was: 'wrong_var' is undefined\n'
       '\n'
       'The error appears to have been in '
       '"playbooks/debugger.yml": line 7, "
       'column 7, but may\n'
       'be elsewhere in the file depending on the exact syntax problem.\n'
       '\n'
       'The offending line appears to be:\n'
       '\n'
       ' tasks:\n'
       '   - name: wrong variable\n'
       '     ^ here\n')

```

© 2025 by Innovation In Software Corporation

31

Check the status of an async job. Register the async task ID, then look at job result variable to confirm it has completed.

## POP QUIZ: DEBUGGING

What is the next step?

- A) Print task arguments
- B) Change variable name
- C) Change variable value
- D) Rerun task?





## POP QUIZ: DEBUGGING

What is the next step?

- A) **Print task arguments**
- B) Change variable name
- C) Change variable value
- D) Rerun task?



This is a bit of a trick question... Different scenarios require different solutions.

DEBUGGING ANSIBLE

Update variable value, and rerun task

```
PLAY *****
...
[192.0.2.10] TASK: wrong variable (debug)> p task.args
(u'data': u'({ wrong_var })')
[192.0.2.10] TASK: wrong variable (debug)> task.args['data'] = '({ var1 })'
[192.0.2.10] TASK: wrong variable (debug)> p task.args
(u'data': '({ var1 })')
[192.0.2.10] TASK: wrong variable (debug)> redo
ok: [192.0.2.10]

PLAY RECAP *****
192.0.2.10 : ok=1 changed=0 unreachable=0 failed=0
```

© 2025 by Innovation In Software Corporation

34

Check the status of an async job. Register the async task ID, then look at job result variable to confirm it has completed.

DEBUGGING ANSIBLE

Print command: prints information about the task

```
[192.0.2.10] TASK: install package (debug)> p task
TASK: install package
[192.0.2.10] TASK: install package (debug)> p task.args
{'name': u'({ pkg_name })'}
[192.0.2.10] TASK: install package (debug)> p task_vars
{'ansible_all_ipv4_addresses': [u'192.0.2.10'],
 u'ansible_architecture': u'x86_64',
 ...
}
[192.0.2.10] TASK: install package (debug)> p task_vars['pkg_name']
u'bash'
[192.0.2.10] TASK: install package (debug)> p host
192.0.2.10
[192.0.2.10] TASK: install package (debug)> p result_result
{'_ansible_no_log': False,
 'changed': False,
 u'failed': True,
 ...
 u'msg': u'No package matching 'not_exist' is available")
```

© 2022 by Innovance IT Software Corporation

## DEBUGGING ANSIBLE



Update args:

```
- hosts: test
  strategy: debug
  vars:
    pkg_name: not_exist
  tasks:
    - name: Install a package
      win_chocolatey: name={{ pkg_name }}
```

### DEBUGGING ANSIBLE

When you run the playbook, the invalid package name triggers an error.  
You can fix the package name by viewing, then updating the module argument.

```
[192.0.2.10] TASK: install package (debug)> p task.args
(u'name': u'({ pkg_name })')
[192.0.2.10] TASK: install package (debug)> task.args['name'] = 'bash'
[192.0.2.10] TASK: install package (debug)> p task.args
(u'name': 'bash')
[192.0.2.10] TASK: install package (debug)> redo
```

### DEBUGGING ANSIBLE

You can also fix the playbook by viewing, then updating the task variables instead of module args.

```
[192.0.2.10] TASK: install package (debug)> p task_vars['pkg_name']
'not_exist'
[192.0.2.10] TASK: install package (debug)> task.vars['pkg_name'] = 'bash'
[192.0.2.10] TASK: install package (debug)> p task_vars['pkg_name']
'bash'
[192.0.2.10] TASK: install package (debug)> update_task
[192.0.2.10] TASK: install package (debug)> redo
```

After updating task variables, you MUST use `update_task` to load the new variables before running the task again with `redo`

## CONTROLLING WHERE TASKS RUN

© 2025 by Innovation In Software  
Corporation



## CONTROLLING WHERE TASKS RUN



By default, Ansible gathers facts and executes all tasks on the machines defined in your playbook.

There are times where delegating tasks, facts etc. to a different machine or group is required.

For example, when updating your webservers, you might need to remove them from a load-balanced pool temporarily.

By delegating the task to localhost, you keep all the tasks within the same play.



## CONTROLLING WHERE TASKS RUN



There are some tasks that can not be delegated:

- `include`
- `add_host`
- `debug`

## DELEGATING TASKS

If you want to perform a task on one host with reference to other hosts, use the `delegate_to` keyword on a task.

This is ideal for managing nodes in a load balanced pool or for controlling outage windows.

```
- hosts: webservers
  serial: 5
  tasks:
    - name: Remove from LB Pool
      command: remove {{ inventory_hostname }}
      delegate_to: 127.0.0.1

    - name: Steps for maintenance
      win_package:
        path: C:\fileshare\app.msi
        state: present

    - name: Add to LB Pool
      command: add {{ inventory_hostname }}
      delegate_to: 127.0.0.1
```

## DELEGATING TASKS

You can also combine `delegate_to` and `command`:

```
- hosts: webservers
  serial: 5
  tasks:
    - name: Remove from LB Pool
      local_action: command remove {{ inventory_hostname }}

    - name: Steps for maintenance
      win_package:
        path: C:\fileshare\app.msi
        state: present

    - name: Add to LB Pool
      local_action: command add {{ inventory_hostname }}
```

© 2025 by Innovation In Software Corporation

44

## POP QUIZ: DISCUSSION

What other uses for `delegate_to` can you think of?



## POP QUIZ: DISCUSSION

What other uses for `delegate_to` can you think of?

- Remove from monitoring
- Send notification (slack, teams, email, Pagerduty)
- Confirm dependent service is online (DB, MQ, Caching)



## DELEGATING TASKS



You can specify additional arguments with the following syntax:

```
tasks:
- name: Send email
  local_action:
    module: community.general.mail
    subject: "Summary Mail"
    to: "{{ mail_receipient }}"
    body: "{{ mail_body }}"
  run_once: True
```

## LOCAL PLAYBOOK



It is possible to have Ansible run a local playbook rather than connecting over SSH.

```
ansible-playbook playbook.yml --connection=local
```

```
- hosts: 127.0.0.1  
  connection: local
```

## POP QUIZ: DISCUSSION

Why use a local playbook?





## POP QUIZ: DISCUSSION

Why use a local playbook?

- Resolve config drift (crontab)
- Bootstrap new host
- OS installer (Kickstart, Anaconda, Foreman)



© 2023 by Innovation Software



